Enhancing practical TAG parsing efficiency by capturing redundancy

Jakub Waszczuk, Agata Savary, Yannick Parmentier

▶ **To cite this version:**

Jakub Waszczuk, Agata Savary, Yannick Parmentier. Enhancing practical TAG parsing efficiency by capturing redundancy. 21st International Conference on Implementation and Application of Automata (CIAA 2016), Jul 2016, Séoul, South Korea. hal-01309598v2

HAL Id: hal-01309598
https://hal.science/hal-01309598v2

Submitted on 3 May 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Enhancing practical TAG parsing efficiency by capturing redundancy*

Jakub Waszczuk[1], Agata Savary[1], and Yannick Parmentier[2]

[1]Université François-Rabelais Tours, Laboratoire d'informatique, France
[2]LIFO - Université d'Orléans, France
(1) first.last@univ-tours.fr (2) first.last@univ-orleans.fr

Abstract. The efficiency of parsing with tree adjoining grammars (TAGs) depends not only on the size of the input sentence but also, linearly, on the size of the input TAG, which can attain several thousands of elementary trees. We propose a factorized, finite-state TAG representation to cope with this combinatorial explosion. The associated parsing algorithm shows a substantial performance gain on a real-size French TAG.

Keywords: parsing, Tree-Adjoining Grammars, grammar compression, finite-state automata, hypergraphs

1 Introduction

High lexicalization and the so-called *extended domain of locality*[1] of TAGs [9], while beneficial for grammar development, are known to lead to very large grammars with up to several thousands of elementary trees [16]. This poses problems of practical nature – parsing algorithms for TAGs are polynomial in the size of the input sentence but also at least linear in the size of the underlying grammar. While many parsing algorithms for speeding up TAG parsing exist, we propose a novel approach in which redundancy is captured by combining and optimizing several previously proposed techniques: grammar flattening, subtree sharing, rule compression into a unique finite-state automaton, and adaptation of parsing inference rules to this representation. Experiments show that these measures lead to a substantial gain in space and time efficiency.

2 Tree Adjoining Grammars

Let Σ and $\mathcal{N}_0$ be disjoint sets of terminal and non-terminal symbols. An *initial tree* (IT) is a tree with non-terminals in non-leaf nodes and terminals/non-terminals in leaf nodes. An *auxiliary tree* (AT) is similar to an IT but it has one

* This work has been supported by the PARSEME European COST Action (IC1207) and by the PARSEME-FR French ANR project (ANR-14-CERA-0001-01).

[1] The former meaning that elementary grammar units are typically attached to one or more lexical items, the latter that many syntactic phenomena can be conveniently represented locally, at the level of individual elementary units.

distinguished leaf (usually marked with an asterisk), called a *foot*, containing the same non-terminal as the root. For instance, in Fig. 2, t_1, t_5 and t_6 are ITs, while t_2, t_3 and t_4 are ATs. A TAG is defined as a tuple $(\Sigma, \mathcal{N}_0, \mathcal{I}, \mathcal{A}, \mathcal{S})$ where $\mathcal{I}$ is the set of elementary initial trees (EITs), $\mathcal{A}$ is the set of elementary auxiliary trees (EATs), and $\mathcal{S}$ is the start non-terminal.

A *derived tree* is created from EITs and EATs by *substitution* and *adjunction*. Given an IT t, and any tree t', substitution replaces a non-terminal leaf l in t' by t provided that labels in l and in t's root are equal. Given an AT t, and any tree t', adjunction replaces t's foot by a subtree t'' of t' and then inserts this modified t in place of t'' in t', provided that the root non-terminals in t and t'' are identical, as shown in Fig. 2. A *derivation tree* keeps track of the operations and the elementary trees (ETs) involved in the creation of a derived tree.

A sequence of terminals obtained by an in-order traversal of a tree t is called a *projection* of t, written $proj(t)$. We also define $proj_A(t)$, specialized to ATs, as a pair of terminal sequences on the left and on the right of the foot node, respectively.

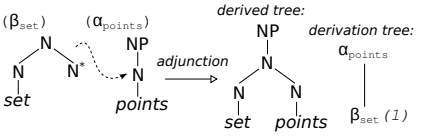


Fig. 1. Adjunction of the tree t_2 to the tree t_5 from Fig. 2

We say that a tree t can be *derived from* a non-trivial subtree[2] t_0 (auxiliary or not) of an ET iff (i) tree t can be derived from the grammar extended with t_0 as an ET and (ii) a derivation tree d of t exists such that t_0 occurs in d's root and, unless t_0 is already part of the grammar, nowhere else in d. We will also say that a non-auxiliary subtree t_0 of an ET is *recognized* over a span (i, l) of the input sentence s iff a tree t can be derived from t_0 such that $proj(t) = s_{(i,l)}$, where $s_{(i,l)}$ is a part of sentence s containing its words between positions i and l. Similarly, we will say that an auxiliary subtree t_0 of an ET is recognized over a span (i, j, k, l) iff a tree t can be derived from t_0 such that $proj_A(t) = (s_{(i,j)}, s_{(k,l)})$.

3 Grammar factorization

Consider the sentence in example (1) and the toy lexicalized TAG (LTAG) containing trees t_1,...,t_6 from Fig. 2 covering several competing interpretations for the two initial words.

(1) **Set points** in tennis belong to official scoring.

The IT t_1 represents *set* as a phrasal verb in imperative mode taking a direct object and a prepositional complement governed by *in*. ATs t_2, t_3 and t_4 consider *set* as a nominal, adjectival and participle modifier of a head noun, respectively. In the IT t_5 *points* is a nominal phrase, while t_6, having two terminals, corresponds to the idiomatic interpretation of *set points* as an NN compound.

[2] In the rest of this paper, by *subtree* we mean a non-trivial (of height > 0) subtree, unless explicitly stated otherwise.

3.1 Grammar flattening with subtree sharing

We propose to represent each ET as a set of flat production rules, so that common subtrees are shared (cf. Fig.2). Each non-terminal from an internal (non-root and non-leaf) node receives a unique index, and each non-leaf node together with its children yields a production rule. E.g., nodes *VP* and *PP* with their children in t_1 yield the rules $VP_1 \to V_2\ NP\ PP_3$, $PP_3 \to P_4\ NP$, respectively. Additionally, each node on the spine of an AT is marked by an asterisk, e.g., the root of t_2 becomes N^* in the head of the rule $N^* \to N_5 N^*$.

Note also that the non-terminal N, occurring twice in t_6, yields two different non-terminals N_0 and N_5 in order to prevent non-compatible rule combinations. For instance, we should not admit an NN-compound *points set*, which would be admitted if these two N terminals were not distinguished. Note, however, also that as soon as some subtrees are common for different grammar trees, the indexed non-terminals, and consequently the target rules, can be shared. For example, the nominal interpretations of *set* and *points* common for t_2, t_5 and t_6 can be shared via the common production rules $N_5 \to set$ and $N_0 \to points$.

In what follows, we refer to such a grammar conversion as *flattening with subtree sharing* (FSS), and to the conversion result as an *FSS grammar* (FSSG).

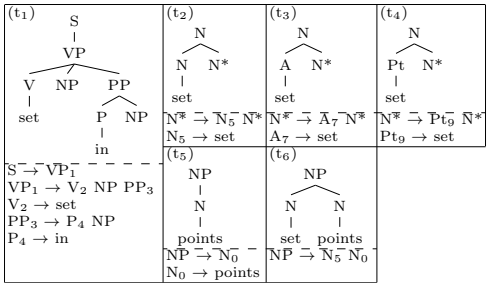


Fig. 2. A toy LTAG grammar and its FSSG.

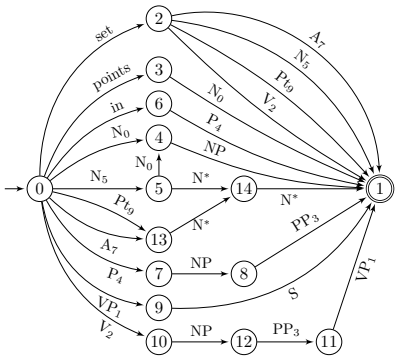


Fig. 3. Compression of the FSSG from Fig.2 into an FSSA.

Formally, the FSSG constructed from a TAG $G = (\Sigma, \mathcal{N}_0, \mathcal{I}, \mathcal{A}, \mathcal{S})$ is a set of production rules $\alpha \in \mathcal{N} \times (\mathcal{N} \cup \Sigma)^+$ where the first and the second component represent the head and the non-empty body of the rule, respectively. $\mathcal{N}_0$ is the set of FSSG non-terminals, i.e. triples $X \in \mathcal{N}_0 \times (\mathbb{N} \cup \{-\}) \times \{-, *\}$ where '$-$' indicates that the corresponding value is unbound. Internal nodes are marked with unique identifiers from the set of natural numbers $\mathbb{N}$. A non-terminal $(x, u, a) \in \mathcal{N}$ is alternatively written as x^a_u and unbounded values $(-)$ are ignored. For example, $(N, -, *)$ is equivalent to N^*, $(V, 2, -)$ to V_2 and $(NP, -, -)$ to NP.

The FSS conversion determines a bijection R_0 between non-terminals originating from internal nodes ($X \in \mathcal{N}_0 \times \mathbb{N} \times \{-, *\}$) and proper subtrees of ETs. A subtree common to several ETs (e.g., the subtree rooted at N dominating *set* in trees t_2 and t_6 in Fig. 2) is represented, in the FSSG, by a single non-terminal (here: N_5). We define a 1-to-many correspondence R between non-terminals ($X = (x, u, a) \in \mathcal{N}$) and TAG subtrees as an extension of this bijection:

$$R(X) = \begin{cases} \{R_0(X)\} & \text{if } u \neq - \\ \mathcal{I}|_x & \text{if } (u, a) = (-, -) \\ \mathcal{A}|_x & \text{if } (u, a) = (-, *) \end{cases} \quad (2)$$

where $\mathcal{I}|_x$ and $\mathcal{A}|_x$ are the sets of all EITs and all EATs, respectively, rooted at $x \in \mathcal{N}_0$. E.g., in Fig. 2, $R(NP) = \{t_5, t_6\}$ and $R(N^*) = \{t_2, t_3, t_4\}$.

3.2 Automaton-based grammar compression

Despite subtree sharing applied to the FSSG in Fig. 2, it still shows some degree of redundancy: the terminal *set* constitutes the body of 4 rules (headed by V_2, N_5, A_7 and Pt_9), the non-terminal *NP* occurs in the head of 2 rules, and the spine non-terminal N^* appears in the head and in the suffix of 3 rules. This observation leads to the idea of representing the FSSG as a minimal deterministic finite-state automaton (DFSA), called here *FSSA*, as shown in Fig. 3. The FSSA's alphabet consists of terminals and non-terminals of the FSSG rules. Each path represents the right-hand side of a rule followed by its head.[3] For instance, the bottom path, traversing nodes 0, 10, 12, 11 and 1, represents the rule $VP_1 \rightarrow V_2\ NP\ PP_3$. In this representation redundancy is largely avoided: the terminal *set* and the head non-terminals *NP* and N^*, are represented by unique transitions $(0, set, 2)$, $(4, NP, 1)$ and $(14, N^*, 1)$, respectively. Additionally, transition $(13, N^*, 14)$ is shared by the suffixes of rules $N^* \rightarrow A_7 N^*$ and $N^* \rightarrow Pt_9 N^*$.

In what follows we extend the notion of an FSSA-based grammar compression into the case when the grammar rules are possibly represented as a *set* of FSSAs (with disjoint sets of node identifiers), according to the particular variant of the compression technique. For instance, in [12] all grammar rules having the same head non-terminal are compressed into a separate DFSA. One of the versions of our parser tested in Sec. 5 implements a similar compression idea.

For a grammar represented as a set of FSSAs, and for any state q therein, let $P(q)$ be a set of sequences of labels leading from an initial state to q. For instance, in Fig. 3, $P(14) = \{N_5 N^*, Pt_9 N^*, A_7 N^*\}$. Note that if q is non-final, sequences in $P(q)$ correspond to prefixes of rules' bodies. In particular, $P(q) \in (N \cup T)^*$.

4 Parser

We propose two Earley-style [6] bottom-up TAG parsing algorithms. The first one, called an *FSS parser*, is inspired by [14], and differs from this seminal work

[3] Head non-terminals are distinguished from others, which is neglected in Fig. 3.

in that it uses an FSSG instead of the original TAG and ignores prediction. The other one, called an *FSSA parser* and inspired by [12], is an extension of the FSS parser in that it uses the FSSG compressed into FSSAs. In both algorithms parsing can be seen, after [11], as a dynamic construction of a *hypergraph* [8] whose nodes are parsing chart items and whose hyperarcs represent applications of inference rules. The hypergraph representation facilitates comparisons between the two algorithms, and time efficiency estimations (the number of elementary parsing steps can be approximated by the number of hyperarcs). It also provides a compressed representation of all the derived trees for a given input sentence.

4.1 FSS parser

Fig. 4 shows the hypergraph created while parsing the two initial words of sentence (1) by the FSS parser with the FSSG from Fig. 2. Due to space constraints, we do not formally define the inference rules of the FSS parser here. They can be seen as simplified versions of those defined in Sec. 4.4. Each item contains a dotted rule and the span over which the symbols to the left of the dot have been parsed. E.g., the hyperarc leading from $(N_5 \rightarrow \bullet set, 0, 0)$ to $(N_5 \rightarrow set\bullet, 0, 1)$ means that the terminal *set* has been recognized from position 0 to 1. The latter item can be combined with $(NP \rightarrow \bullet N_5 N_0, 0, 0)$ yielding $(NP \rightarrow N_5 \bullet N_0, 0, 1)$, etc. The sentence s has been parsed if a goal item has been reached (spanning from 0 to $|s|$, with a rule headed by $(S, -, -)$ and terminated by a dot).

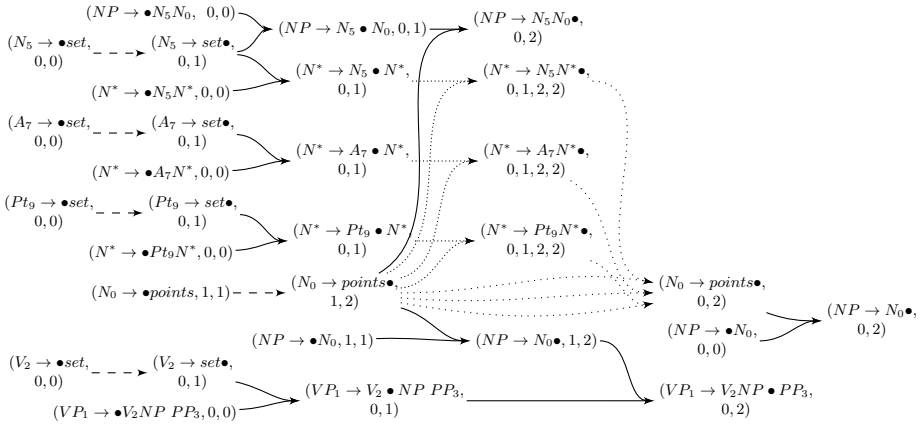


Fig. 4. Hypergraph created by the FSS parser while parsing the substring *set points* with the FSSG from Fig. 2. The dashed and plain hyperarcs roughly correspond to scanner and completer operations in a CFG Earley parser. The densely and loosely dotted hyperarcs represent novel inference rules: foot adjoin and root adjoin.

Items whose spans contain 4 integers (i_1, i_2, i_3, i_4) result from the FSS-based inference rules related to adjunction: i_1 and i_4 represent the whole span of the

recognized sequence, while i_2 and i_3 indicate the *gap*, i.e., the part of the sequence matched by the foot node of an AT. For instance, the hyperarc leading from $(N^* \rightarrow N_5 \bullet N^*, 0, 1)$ and $(N_0 \rightarrow points\bullet, 1, 2)$ to $(N^* \rightarrow N_5 N^* \bullet, 0, 1, 2, 2)$ puts forward an adjunction hypothesis. The noun *points* has been recognized over span $(1, 2)$, and *set* recognized over $(0, 1)$ might later be adjoined to it as a modifier. Thus, *points* will fill the gap (from 1 to 2) corresponding to the foot node N^* in the body of rule $N^* \rightarrow N_5 N^*$ (stemming from tree t_2). Note further that the combination of items $(N^* \rightarrow N_5 N^* \bullet, 0, 1, 2, 2)$ and $(N_0 \rightarrow points\bullet, 1, 2)$ yields $(N_0 \rightarrow points\bullet, 0, 2)$, which corresponds to stage 1 of the adjunction (see Sec. 2). Stage 2 is then represented by the hyperarc leading to $(NP \rightarrow N_0 \bullet, 0, 2)$.

4.2 FSSA parser

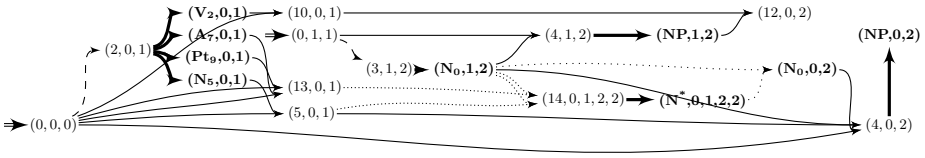


Fig. 5. Hypergraph representing the chart parsing of the substring *set points* with the FSSA from Fig. 3. The double, plain, thick, dashed, densely dotted and loosely dotted hyperarcs represent axioms, pseudo substitution, deactivate, scan, foot adjoin and root adjoin inference rules, respectively (see Sec. 4.4). Passive states are highlighted in bold.

The idea behind grammar compression is not only space efficiency but also reducing parsing time [12]. The latter is based on the observation that, whenever bodies of some flat rules share common prefixes and/or suffixes (which is in close relation to sharing sub-paths in the FSSA), partial parsing results can be shared for them. Another related fact is that, for a given position of the dot in a flat dotted rule, the history of the parsing on the left-hand side of the dot does not influence the future parsing on the right-hand side of the dot. Therefore, the position of the dot in a rule can be nicely represented by the FSSA state achieved while parsing the rule, whatever the path which led us to this state.

These observations may lead to a substantial reduction of the parsing hypergraph, as shown in Fig. 5. Here, dotted rules in the hypergraph items from Fig. 4 are replaced by states of the FSSA from Fig. 3 (the resulting items are called *active*). Firstly, all 9 initial items (i.e., having the dot at the beginning of their rules' bodies) over span $(0, 0)$ in Fig. 4, e.g., $(N_5 \rightarrow \bullet set, 0, 0)$, $(NP \rightarrow \bullet N_5 N_0, 0, 0)$, etc. – are replaced by a unique item $(0, 0, 0)$ in Fig. 5 due to the fact that they all share the same (empty) prefix on the left-hand side of the dot, and the same span. The 10^{th} remaining initial item $(N_0 \rightarrow \bullet points, 1, 1)$ is replaced by $(0, 1, 1)$. Further, rules having dots inside their bodies are replaced by FSSA states, for instance items $(N^* \rightarrow A_7 \bullet N, 0, 1)$ and $(N^* \rightarrow Pt_9 \bullet N, 0, 1)$ are replaced by

the unique item $(13, 0, 1)$ since their prefixes A_7 and Pt_9 lead to the same state 13. Finally, complete items (i.e., having the dot at the end of the rule), are replaced by two items, the one containing the arrival state, and the other (called a *passive* item) in which the state is replaced by the head of the fully recognized rule. For instance, items $(N^* \rightarrow N_5 N^* \bullet, 0, 1, 2, 2)$, $(N^* \rightarrow A_7 N^* \bullet, 0, 1, 2, 2)$ and $(N^* \rightarrow Pt_9 N^* \bullet, 0, 1, 2, 2)$ are merged into one active item $(14, 0, 1, 2, 2)$ since they share the same arrival state 14 and span. This item is then followed by a passive item $(N^*, 0, 1, 2, 2)$. The goal item is $(S, 0, |s|)$.

4.3 Items

Let $s = s_0 s_1 \dots s_{n-1}$ be the input sentence and $Pos(s) = \{0, \dots, n\}$ the set of positions between the words in s, before s_0 and after s_{n-1}. We define two kinds of items. A **passive item** is a tuple (X, i, j, k, l) where: $X \in \mathcal{N}$, $i, l \in Pos(s)$, $j, k \in Pos(s) \cup \{-\}$, $i \leq l$, and $i \leq j \leq k \leq l$ if $(j, k) \neq (-, -)$. Item (X, i, j, k, l) asserts that X can be *matched* over the span (i, j, k, l), where (i, l) and (j, k) denote the whole span of a matched sequence and the gap, respectively. Formally, a passive item $(X, i, -, -, l)$, or (X, i, l) for short, asserts that an IT $t \in R(X)$, a subtree of an ET in G, can be recognized (cf. Sec. 2) over the span (i, l). E.g., item $(N_0, 1, 2)$ in Fig. 5 indicates that *points* in sentence (1) can be a noun by the subtree rooted at N in t_5 and t_6 in Fig. 2. A passive item (X, i, j, k, l) where $(j, k) \neq (-, -)$ and $X = (x, u, a)$ asserts that (i) an AT $t \in R(X)$, a subtree of some ET in G, can be recognized over (i, j, k, l), and (ii) a subtree t' of an ET[4], with $x \in \mathcal{N}_0$ in its root, can be recognized over (j, k). Thus, the item $(N^*, 0, 1, 2, 2)$ in Fig. 5 means that *set* can be a modifier adjoined to the noun *points*. Here: $t \in \{t_2, t_3, t_4\}$ and t' is the subtree rooted at N in t_5 and t_6. An **active item** is a tuple (q, i, j, k, l), where i, j, k, and l specify the span, as previously, and q is a state in one of the underlying FSSAs. An active item (q, i, j, k, l) asserts that there exists a (not necessarily proper) prefix $\omega \in P(q)$ (of a grammar rule's body) which can be matched over (i, j, k, l), i.e., that the individual elements of ω can be consecutively matched over the adjacent spans of the input sentence, together spanning over (i, l), and that, if $(j, k) \neq (-, -)$, one of the elements of ω, marked with an asterisk, is matched against the item's gap (j, k). E.g., $(12, 0, 2)$ and $(14, 0, 1, 2, 2)$ in Fig. 5 correspond to matching *set* *points* with $\omega = V_2 NP$ and $\omega \in \{N_5 N^*, Pt_9 N^*, A_7 N^*\}$, respectively.

4.4 Inference rules

We now formally specify the FSSA parser using the deductive framework [15]. As shown in Tab. 1, each of the inference rules, whose applications correspond to hyperarcs in the parsing hypergraph, takes zero, one or two chart items on input (*premises*, presented above the horizontal line) and yields a new item (*conclusion*, presented below the line) to be added to the chart if the conditions given on the right-hand side are met. The **axiom** rule (AX, cf. the double hyperarcs with

[4] t' must not be an EAT (see the *root adjoin* inference rule in Sec. 4.4 for explanations).

AX:	$\frac{}{(q_0,i,-,-,i)}$	$i \in Pos(s) \setminus \{n\}$	PS:	$\frac{(q,i,j,k,l) \quad (X,l,-,-,l')}{(\delta(q,X),i,j,k,l')}$	$\delta(q,X)$ defined
SC:	$\frac{(q,i,j,k,l)}{(\delta(q,s_l),i,j,k,l+1)}$	$\delta(q,s_l)$ defined	FA:	$\frac{(q,i,-,-,l) \quad (X,l,j,k,l')}{(\delta(q,Y),i,l,l',l')}$	$(x,u,a)=X$ $(u,a)\neq(-,*)$ $Y=(x,-,*)$ $\delta(q,Y)$defined
DE:	$\frac{(q,i,j,k,l)}{(X,i,j,k,l)}$	$X \in heads(q)$	IA:	$\frac{(q,i,-,-,l) \quad (X,l,j,k,l')}{(\delta(q,X),i,j,k,l')}$	$\delta(q,X)$ defined $(j,k)\neq(-,-)$
RA:	$\frac{(X,i,j,k,l) \quad (Y,j,j',k',k)}{(Y,i,j',k',l)}$		$(x,-,*) = X, (y,u,a) = Y, (u,a) \neq (-,*), x = y$		

Table 1. Inference rules of the FSSA parser

empty inputs leading to $(0,0,0)$ and $(0,1,1)$ in Fig. 5) fills the initially empty chart with active items representing the claim that any rule α from the FSSG can be used to parse s starting from any non-final position, for each initial state q_0 of one of the FSSAs. The **scan** rule (SC, cf. the dashed hyperarcs in Fig. 5) matches the FSSAs' terminal symbols with words from the input. **Deactivation** (DE, cf. the thick hyperarcs) transforms an active item into the corresponding passive item, based on the q-outgoing head non-terminals, where $heads(q)$ is the set of symbols over transitions (representing rule heads) from state q to final states of the FSSAs. **Pseudo substitution** (PS, cf. the plain hyperarcs) is similar to scan, but instead of matching FSSA terminals against input words, automaton non-terminals are matched against already inferred non-terminals represented by passive items. Pseudo substitution handles regular TAG substitution, i.e., replacing a leaf non-terminal X by an IT rooted by X (cf. the hyperarc leading from $(10,0,1)$ and $(NP,1,2)$ to $(12,0,2)$), as well as matching two adjacent fragments of the same ET (cf. the hyperarc from $(5,0,1)$ and $(N_0,1,2)$ to $(4,0,2)$). The **foot adjoin** rule (FA, cf. the densely dotted hyperarcs) identifies ranges over which adjunction could possibly occur. It ensures that the resulting item is considered only if an elementary (sub)tree, recognized starting from l, and to which the corresponding AT(s) could be adjoined, exists. For the hyperarc from $(5,0,1)$ and $(N_0,1,2)$ to $(14,0,1,2,2)$, we have $X = N_0 = (N,0,-)$, $Y = (N,-,*) = N^*$, $(j,k) = (-,-)$ and $\delta(5,Y) = 14$. The **internal adjoin** rule (IA, with no instance in Fig. 5) combines an elementary (sub)tree, partially recognized over $(i,-,-,l)$, with its spine subtree, recognized starting from position l. Internal adjoin is similar to pseudo substitution but must be handled by a separate rule because the span of gap in the conclusion stems from the passive rather than the active premise. The **root adjoin** rule (RA, cf. the loosely dotted hyperarcs) represents the actual adjoining of a fully recognized EAT t into the root of a recognized subtree t' of an ET. Information that t' is recognized (with a modified span), is preserved in the conclusion and can be reused in order to recognize the full ET of which t' is a part. E.g., for the hyperarc from $(N^*,0,1,2,2)$ and $(N_0,1,2)$ to $(N_0,0,2)$, we have $X = N^* = (N,-,*)$, $Y = N_0 = (N,0,-)$, $x = y = N$, $(u,a) = (0,-)$, $(j',k') = (-,-)$, $t \in \{t_2,t_3,t_4\}$ and t' is the subtree of t_5 rooted at N.[5]

[5] Note that the additional constraint imposed on the modified node is that it must not be a root of an AT ($(u,a) \neq (-,*)$). Otherwise, it would be possible to adjoin

5 Experimental results

We performed experiments on the FrenchTAG meta-grammar [5] compiled into a set of 9043 non-lexicalized ETs. After removing feature structures (not supported by our parser) 3065 unique trees where obtained. Since no compatible lexicon is available, we lexicalized the grammar with part-of-speech (POS) tags. Namely, to each anchor (i.e., the node meant to receive a terminal from an associated lexicon) in each ET a special terminal, containing the same POS value as the anchor, was attached. Thus, we obtained a grammar which models sentences with regular terminals (e.g., *il* 'it', *de* 'of', *qui* 'who') and POS tags (e.g., *v*, *n*, *adj*) interleaved. Such (inevitably, due to the missing lexicon) artificial lexicalization is not fully satisfactory in the context of TAGs, but it gives us an approximate upper bound on the possible gain from our compression-based approach.

Fig. 6(a) shows the total numbers of automaton states and transitions depending on the compression method used to encode the resulting grammar. In the baseline, the grammar is represented as a list of flat rules (encoded as a separate automaton each) but no subtree sharing takes place. With this representation, parsing is roughly equivalent to the Earley-style TAG algorithm [14]. The FSS and FSSA encoding methods were described in Sec. 3.1 and 3.2.

Since treebanks compatible with existing TAGs (especially those generated from metagrammars) are hardly available, parsing evaluation was done on a synthetic corpus. Namely, ∼13000 sentences of length 1 to 30, of up to 500 sentences per length, were used to measure performance in terms of the number of hyperarcs explored while parsing a sentence (deactivate operations are ignored). The results are presented in Fig. 6(b), which includes two additional grammar compression methods similar to those in [12] for CFGs: (i) a trie, in which the list of rules is transformed into a prefix tree instead of a DFSA, (ii) a set of FSSAs, where a separate DFSA is constructed for each rule head.

The results show that the baseline version of the parser is only of a theoretical interest. It requires generating on average more than 4×10^4 hyperarcs even for sentences of length 1 (notably due to the POS-based lexicalization). The FSS parser is already a reasonable choice for parsing short sentences. FSSA compression leads, averaging over sentences of length from 1 to 15, to a farther reduction of ∼24× in terms of the number of visited hyperarcs. Using a set of FSSAs instead of a single FSSA is ∼2.25 times less efficient on average.

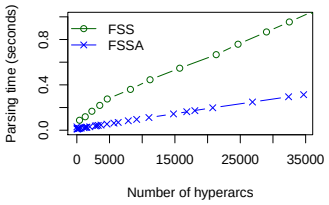
Fig. 6(c) compares the FSS and FSSA parsers in terms of speed. In both versions, parsing time is almost linear w.r.t. the number of generated hyperarcs. However, the FSSA version proves more efficient, most likely due to the number of generated hypernodes which is, consistently, significantly higher in the FSS version (e.g., 95666 hypernodes in FSS against 1193 in FSSA for sentences of length 15). This, in turn, is related to the fact that a large number of (trivial)

---

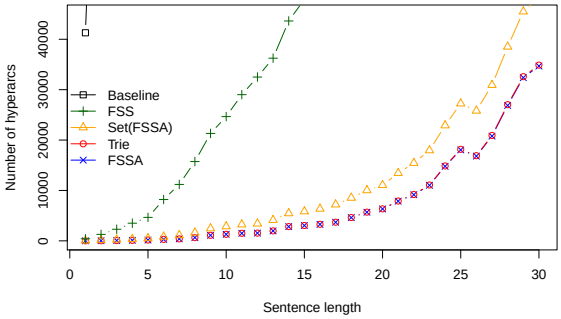
one AT to a root of another *not yet adjoined* AT. We block this derivation path, so that adjunction can only be carried out on top of an AT which has already been adjoined to some particular IT.

	Baseline	FSS	FSSA
States	83421	19387	1192
Transitions	61474	15212	5225

(a)



(c)



(b)

Fig. 6. (a) Results of the compression experiments, (b) Impact of grammar encoding methods on parsing performance, measured as an average number of hyperacs explored by the parser on ∼13000 sentences randomly generated from the FrenchTAG grammar, (c) Average parsing time as a function of the number of generated hyperarcs.

automata is used in the FSS parser, thus a large number of initial states have to be handled by the the **axiom** rule at the very beginning of the parsing process.

Surprisingly, the FSSA compression does not bring significant improvements in comparison to the prefix tree version. This is probably related to the fact that the active/passive distinction already provides a form of suffix sharing – items referring to pre-final states in the prefix tree are automatically transformed into the corresponding passive items. In particular, the number of passive items which can be potentially constructed over a given span equals 1123 in both versions, while the number of potential active items per span diminishes merely from 430 to 301 in the FSSA version. Moreover, due to the left-to-right parsing strategy, prefix sharing impacts parsing performance more visibly than suffix sharing.

6 Related work

A bottom-up Earley-like algorithm based on flattening is one of the TAG parsing schemata proposed in [2]. While, conversely to our approach, it does not allow multiple adjunctions at the same node, it is similar to our baseline algorithm. Our enhancements of this baseline with subtree sharing and grammar FSA-compression substantially influence space and time efficiency (cf. Sec. 5).

FSA-based grammar encoding considerably speeds up CFG parsing [12] but it is not straightforwardly applicable to TAGs (which consist of trees rather than flat rules). It is, however, enabled by the flattening transformation proposed in this paper. Previous proposals of applying FSA-based compression to TAGs are manifold. [10] and [13] describe LR parsers for TAGs, in which predictions are pre-compiled off-line into an FSA. Each state of this FSA is a set of dotted production rules closed under prediction. Thus, the FSA represents the parser, while in our approach the FSSA represents the grammar (and the inferences rules of the parser are adapted to this representation).

Another automata-based solution for LTAGs and related lexicalized formalisms has been proposed by [7, 4]. The traversal of an ET, starting from its anchor (lexical unit), is represented there as an automaton. Sets of trees attached to common anchors are then converted to automata, merged and minimized using standard techniques. As a result, structure sharing occurs only within tree families, while in our solution all ETs are represented with a single automaton which provides sharing between rules assigned to different lexical units. Another potential advantage of our solution lies in the subtree-sharing it enables, which allows different rules – even when represented by completely different paths in the automaton – to share common middle elements if these middle elements represent common subtrees. Finally, our method can be used for TAGs in general, not only for lexicalized TAGs. [4] report state-level compression ratios equal to 18 for *come*, 18.2 for *break*, and 30 for *give*, over a lexicalized English grammar. We converted the XTAG grammar [1] into an FSSA, obtaining a global, state-level compression of 22.7 (10751 states in the baseline representation vs. 472 in the FSSA). It is, however, difficult to compare these numbers: (i) their grammar is considerably larger than XTAG, (ii) they did not report the compression ratio over the entire grammar, (iii) they use one automaton per input word While they did not measure the impact of their encoding on parsing performance, we believe that our FSSA-based solution is more scalable w.r.t. the input length.

[16] proposes a method of grammar compression directly at stage of its definition. A linguist uses a formal language including factoring operators (e.g., disjunctions over tree fragments, Kleene-star-alike repetitions, optional or shuffled fragments, etc.) and the resulting grammar is then converted into a Logic Push-Down Automaton for parsing. The price to pay for this highly compact resource is its high potential overgeneration. Moreover, grammar description and parsing are not separated, hence large unfactorized TAGs can be hardly coped with. Our solution abstracts away from how the TAG is represented, compression is automatic and the FSSA is strongly equivalent to the original TAG.

Linear indexed grammars (LIGs) compare to our grammar flattening in that they contain flat production rules and are weakly equivalent to TAGs [10]. However, LIGs are more generic than TAGs, thus more specialized and efficient parsers can be potentially designed for TAGs [3]. Also, the TAG-to-LIG conversion does not preserve the extended domain of locality (EDL) ensured by TAGs, which is for us an eliminating criterion. Namely, in future we wish our parser to be driven by the knowledge about possible occurrences of multi-word expressions [17], whose elegant representation in TAGs is precisely due to the EDL property.

7 Conclusions

Our contribution is to design a parsing architecture coping with large TAGs (notably produced from metagrammars). We build on previous work so as to capture redundancy: (i) we flatten TAGs, (ii) we share common subtrees, (iii) we compress the flat grammar into an FSA, (iv) we adapt an Earley-based algorithm to this representation, (v) we show the influence of these steps on the parsing

efficiency. To the best of our knowledge this is the first attempt to combine all these steps within one framework. Our parser and evaluation corpus are available under open licenses.[6] This solution does not affect the theoretical complexity of TAG parsing but it greatly improves the practical parsing performance.

References

1. Katya Alahverdzhieva. XTAG using XMG, 2008. Master Thesis, Nancy Université.
2. Miguel Alonso, David Cabrero, Eric Villemonte de la Clergerie, and Manuel Vilares Ferro. Tabular algorithms for TAG parsing. In *EACL 1999*, pages 150–157, 1999.
3. Miguel A. Alonso, Éric Villemonte de La Clergerie, Vítor J. Diaz, and Manuel Vilares. *Relating Tabular Parsing Algorithms for LIG and TAG*, volume 23 of *Text, Speech and Language Technology*, pages 157–184. Kluwer Ac. Publ., 2004.
4. John Carroll, Nicolas Nicolov, Olga Shaumyan, Martine Smets, and David Weir. Grammar Compaction and Computation Sharing in Automaton-based Parsing. In *Proceedings of the TAPD'98 Workshop*, pages 16–25, Paris, France, 1998.
5. Benoit Crabbé. *Représentation informatique de grammaires d'arbres fortement lexicalisées : le cas de la grammaire d'arbres adjoints.* PhD thesis, Université Nancy 2, 2005.
6. Jay Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, February 1970.
7. Roger Evans and David Weir. Automaton-based Parsing for Lexicalized Grammars. In *Proceedings of the IWPT'97 Workshop*, pages 66–76, Boston, MA, 1997.
8. Giorgio Gallo, Giustino Longo, Stefano Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, April 1993.
9. A. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, pages 69–123. Springer, 1997.
10. Laura Kallmeyer. *Parsing Beyond Context-Free Grammars.* Springer, 2010.
11. Dan Klein and Christopher D. Manning. Parsing and hypergraphs. In *Proceedings of the IWPT'01 Workshop.* Tsinghua University Press, 2001.
12. Dan Klein and Christopher D. Manning. Parsing with Treebank Grammars: Empirical Bounds, Theoretical Models, and the Structure of the Penn Treebank. In *Proceedings of ACL'01*, pages 338–345, 2001.
13. Carlos A. Prolo. Fast LR Parsing Using Rich (Tree Adjoining) Grammars. In *Proceedings of EMNLP'02*, pages 103–110, 2002.
14. Yves Schabes. Left to Right Parsing of Lexicalized Tree Adjoining Grammars. *Computational Intelligence*, 10(4):506–524, 1994.
15. Stuart Shieber, Yves Schabes, and Fernando Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1):3–36, 1995.
16. Éric Villemonte de La Clergerie. Building factorized TAGs with meta-grammars. In *Proceeding of the TAG+10 Conference*, 2010.
17. Jakub Waszczuk and Agata Savary. Towards a MWE-driven A* parsing with LTAGs. In *PARSEME 6th general meeting*, Struga, FYR Macedonia, 2016.

[6] `https://github.com/kawu/partage4xmg/tree/0.1`